```python
import os
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.regularizers import l2


dataset_dir = 'leukemia/Original'
class_names = ['Benign', 'Pre', 'Pro', 'Early']

images = []
labels = []

for class_name in class_names:
    class_dir = os.path.join(dataset_dir, class_name)
    for img_name in os.listdir(class_dir):
        img_path = os.path.join(class_dir, img_name)
        img = load_img(img_path, target_size=(224, 224))  # Resize to match input size of VGG/ResNet
        img = img_to_array(img)
        img = preprocess_input(img)
        images.append(img)
        labels.append(class_name)

images = np.array(images)
labels = np.array(labels)


label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)


X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=None, stratify=labels)
```

```
train_datagen = ImageDataGenerator()

test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow(X_train, y_train, batch_size=128)
test_generator = test_datagen.flow(X_test, y_test, batch_size=128)
```

ResNet50 Model Training

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(class_names), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

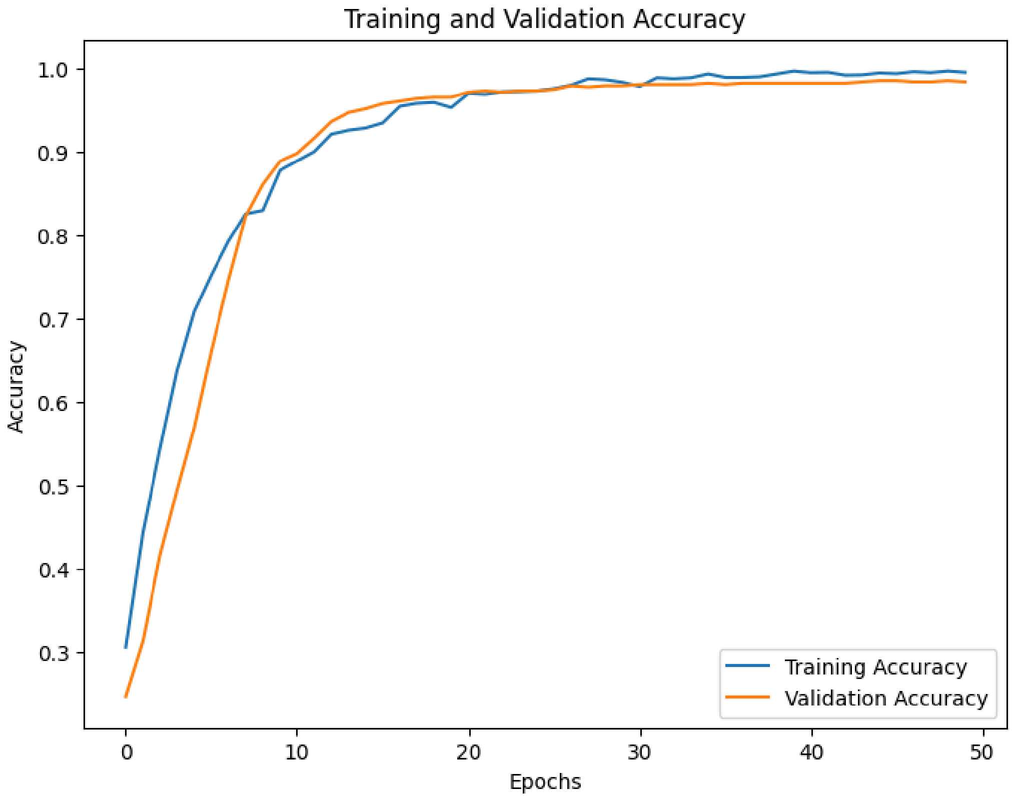
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
model.compile(optimizer=Adam(learning_rate=1e-6),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(train_generator,
                    epochs=50,
                    validation_data=test_generator,
                    callbacks=[early_stopping])
```

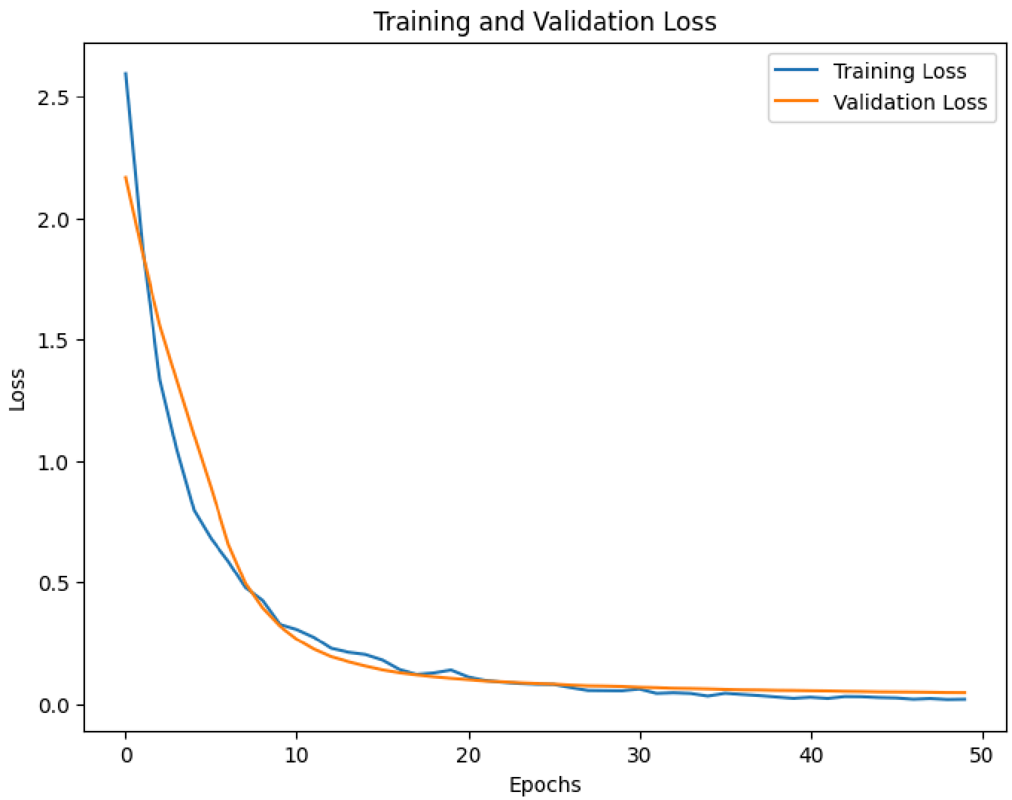
```
Epoch 1/50
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__in
  self._warn_if_super_not_called()
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1733723791.477504      66 service.cc:145] XLA service 0x7d22e4003950 initialized for platform CUDA (this does not guarantee that XLA will be used). D
I0000 00:00:1733723791.477572      66 service.cc:153]   StreamExecutor device (0): Tesla P100-PCIE-16GB, Compute Capability 6.0
2024-12-09 05:56:47.623195: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[128,1024,14,14]{3,2,1,0}, u8[0]{0}
2024-12-09 05:56:48.544450: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 1.921382707s
Trying algorithm eng0{} for conv (f32[128,1024,14,14]{3,2,1,0}, u8[0]{0}) custom-call(f32[128,2048,7,7]{3,2,1,0}, f32[2048,1024,1,1]{3,2,1,0}), window={size=1x1
2024-12-09 05:56:51.013014: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[128,512,28,28]{3,2,1,0}, u8[0]{0})
2024-12-09 05:56:51.884554: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 1.87172351s
Trying algorithm eng0{} for conv (f32[128,512,28,28]{3,2,1,0}, u8[0]{0}) custom-call(f32[128,1024,14,14]{3,2,1,0}, f32[1024,512,1,1]{3,2,1,0}), window={size=1x1
2024-12-09 05:56:54.452511: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[128,256,56,56]{3,2,1,0}, u8[0]{0})
2024-12-09 05:56:55.158751: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 1.706377628s
Trying algorithm eng0{} for conv (f32[128,256,56,56]{3,2,1,0}, u8[0]{0}) custom-call(f32[128,512,28,28]{3,2,1,0}, f32[512,256,1,1]{3,2,1,0}), window={size=1x1 st
I0000 00:00:1733723835.817989      66 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
21/21 ━━━━━━━━━━━━━━━━━━━━ 118s 2s/step - accuracy: 0.2867 - loss: 2.7405 - val_accuracy: 0.2469 - val_loss: 2.1655
```

```
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 555ms/step - accuracy: 0.4088 - loss: 2.0137 - val_accuracy: 0.3129 - val_loss: 1.8502
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 553ms/step - accuracy: 0.5286 - loss: 1.3552 - val_accuracy: 0.4156 - val_loss: 1.5572
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 556ms/step - accuracy: 0.6218 - loss: 1.0973 - val_accuracy: 0.4939 - val_loss: 1.3286
Epoch 5/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 553ms/step - accuracy: 0.7016 - loss: 0.8036 - val_accuracy: 0.5690 - val_loss: 1.1045
Epoch 6/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 555ms/step - accuracy: 0.7390 - loss: 0.7224 - val_accuracy: 0.6610 - val_loss: 0.8874
Epoch 7/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 554ms/step - accuracy: 0.7800 - loss: 0.6148 - val_accuracy: 0.7454 - val_loss: 0.6566
Epoch 8/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 554ms/step - accuracy: 0.8247 - loss: 0.5028 - val_accuracy: 0.8221 - val_loss: 0.4957
Epoch 9/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 557ms/step - accuracy: 0.8266 - loss: 0.4289 - val_accuracy: 0.8604 - val_loss: 0.3945
Epoch 10/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 551ms/step - accuracy: 0.8785 - loss: 0.3265 - val_accuracy: 0.8880 - val_loss: 0.3204
Epoch 11/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 554ms/step - accuracy: 0.8924 - loss: 0.2972 - val_accuracy: 0.8972 - val_loss: 0.2662
Epoch 12/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 556ms/step - accuracy: 0.8956 - loss: 0.2797 - val_accuracy: 0.9156 - val_loss: 0.2253
Epoch 13/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 553ms/step - accuracy: 0.9256 - loss: 0.2230 - val_accuracy: 0.9356 - val_loss: 0.1940
Epoch 14/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 555ms/step - accuracy: 0.9217 - loss: 0.2191 - val_accuracy: 0.9463 - val_loss: 0.1738
Epoch 15/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 553ms/step - accuracy: 0.9245 - loss: 0.2198 - val_accuracy: 0.9509 - val_loss: 0.1558
Epoch 16/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 554ms/step - accuracy: 0.9404 - loss: 0.1716 - val_accuracy: 0.9571 - val_loss: 0.1399
Epoch 17/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 553ms/step - accuracy: 0.9532 - loss: 0.1472 - val_accuracy: 0.9601 - val_loss: 0.1279
Epoch 18/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 554ms/step - accuracy: 0.9541 - loss: 0.1328 - val_accuracy: 0.9632 - val_loss: 0.1192
Epoch 19/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 555ms/step - accuracy: 0.9589 - loss: 0.1216 - val_accuracy: 0.9647 - val_loss: 0.1117
Epoch 20/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 553ms/step - accuracy: 0.9543 - loss: 0.1359 - val_accuracy: 0.9647 - val_loss: 0.1052
Epoch 21/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 13s 554ms/step - accuracy: 0.9734 - loss: 0.1063 - val_accuracy: 0.9709 - val_loss: 0.0998
```

```
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```




```
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```




```
model.save('resnet50.h5')
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

# Print the results
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 47ms/step - accuracy: 0.9787 - loss: 0.0540
Test Loss: 0.0468
Test Accuracy: 0.9831
```

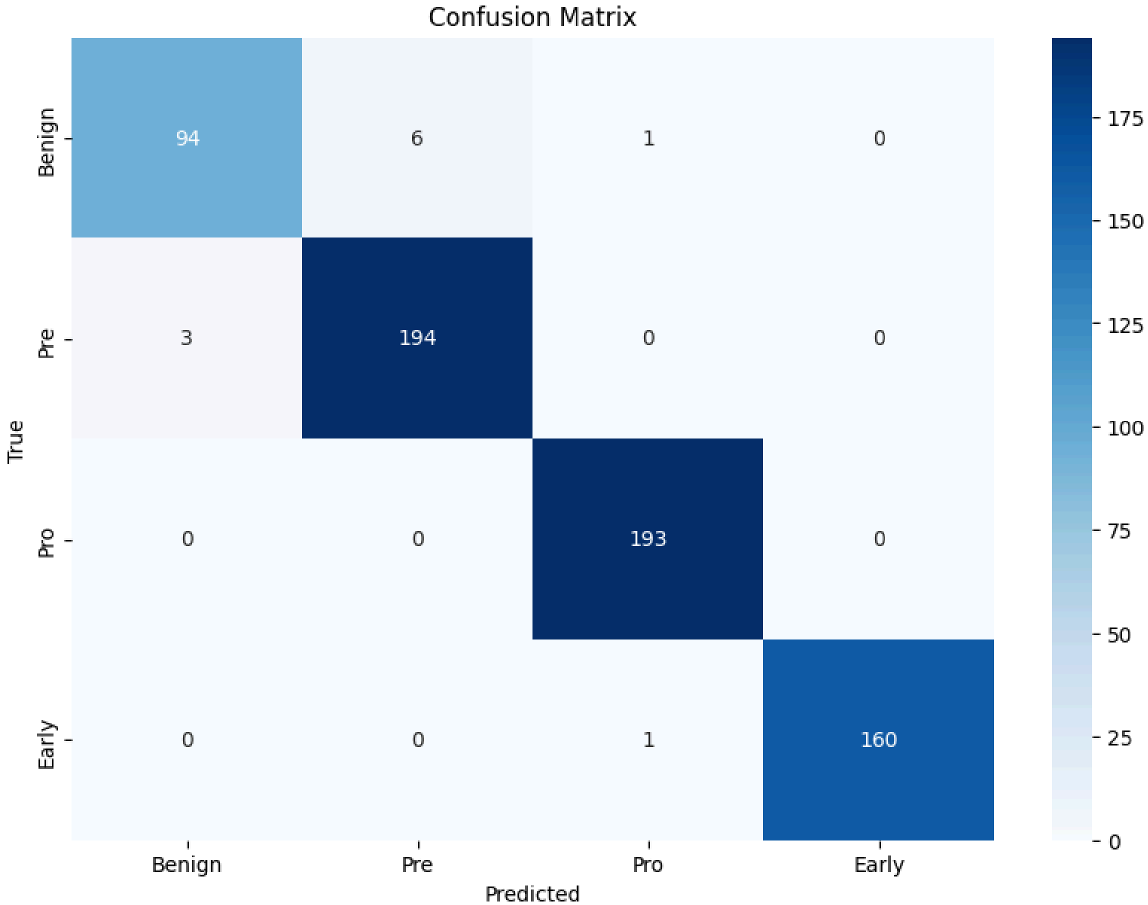
ResNet50 Evaluation Metrics

```
from tensorflow.keras.models import load_model

# Load the model
model = load_model('resnet50.h5')
```

```
from sklearn.metrics import classification_report

# Predict classes for the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Classification report
print(classification_report(y_test, y_pred_classes, target_names=class_names))
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 157ms/step
              precision    recall  f1-score   support

      Benign       0.97      0.93      0.95       101
         Pre       0.97      0.98      0.98       197
         Pro       0.99      1.00      0.99       193
       Early       1.00      0.99      1.00       161

    accuracy                           0.98       652
   macro avg       0.98      0.98      0.98       652
weighted avg       0.98      0.98      0.98       652
```

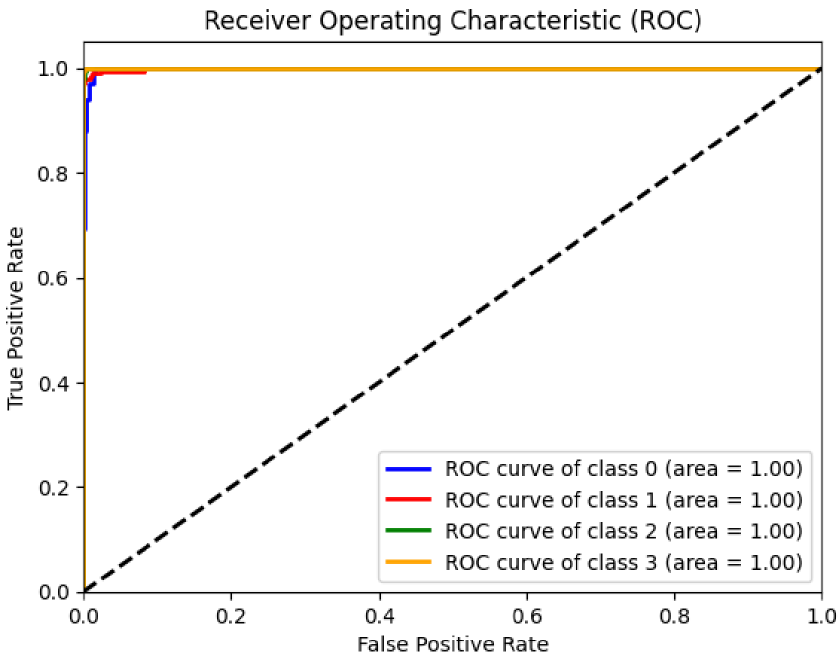
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```




```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Convert y_test to one-hot encoding
y_test_one_hot = label_binarize(y_test, classes=[0, 1, 2, 3])  # Adjust classes as needed

# Predict probabilities
y_prob = model.predict(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_test_one_hot[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Plot ROC curves
plt.figure()
colors = ['blue', 'red', 'green', 'orange']  # Adjust for your classes

for i, color in zip(range(len(class_names)), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step
```



EfficientNetB0 Model Training

```
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(class_names), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

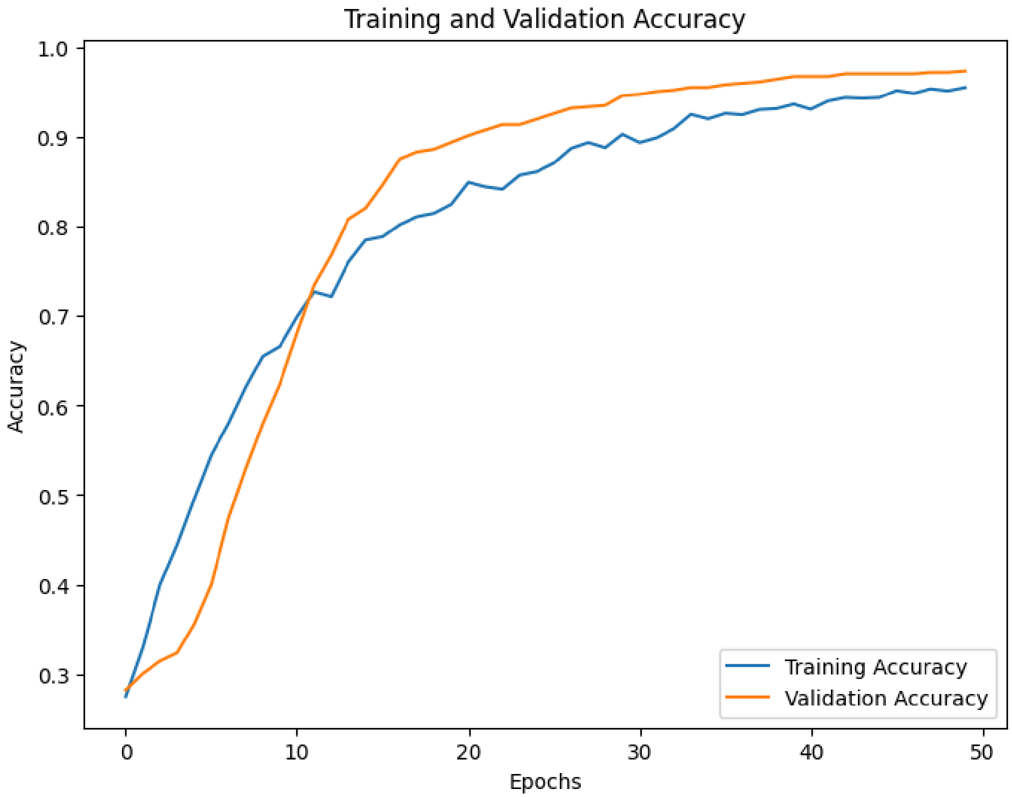
```
model.compile(optimizer=Adam(learning_rate=1e-6),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(train_generator,
                    epochs=50,
                    validation_data=test_generator,
                    callbacks=[early_stopping])
```

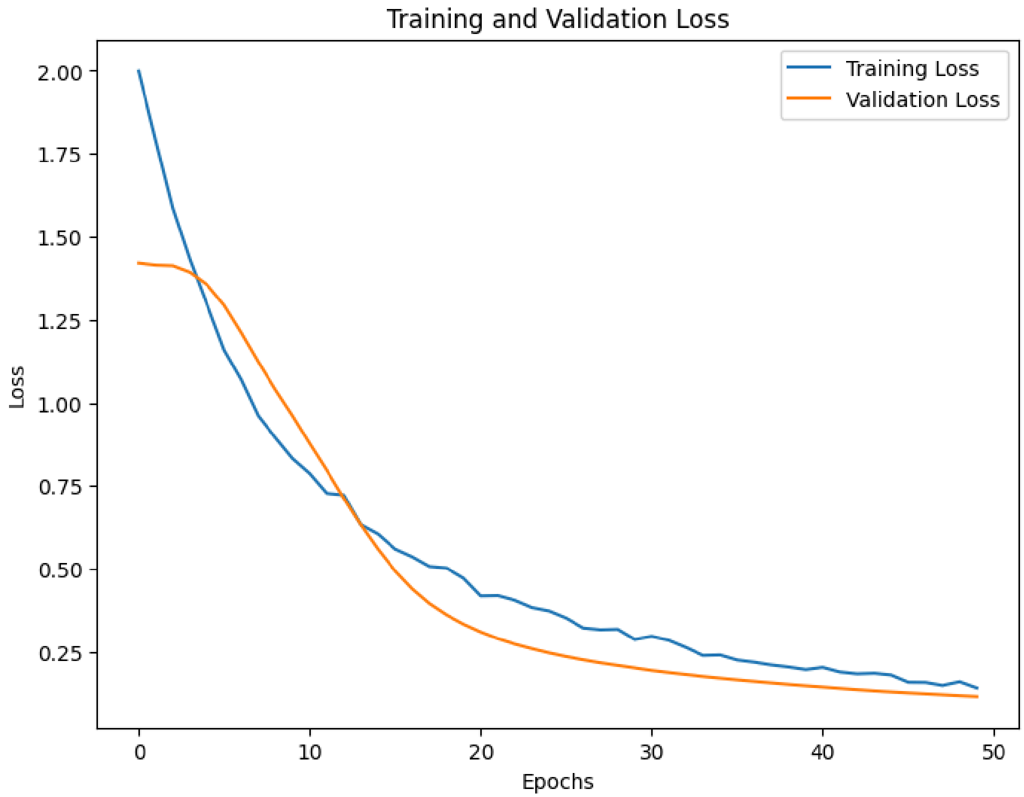
```
Epoch 1/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 149s 3s/step - accuracy: 0.2623 - loss: 2.0601 - val_accuracy: 0.2822 - val_loss: 1.4204
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 440ms/step - accuracy: 0.3157 - loss: 1.8489 - val_accuracy: 0.3006 - val_loss: 1.4143
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 437ms/step - accuracy: 0.3844 - loss: 1.6415 - val_accuracy: 0.3144 - val_loss: 1.4125
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 449ms/step - accuracy: 0.4443 - loss: 1.4387 - val_accuracy: 0.3236 - val_loss: 1.3929
Epoch 5/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 439ms/step - accuracy: 0.4915 - loss: 1.3346 - val_accuracy: 0.3558 - val_loss: 1.3556
Epoch 6/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 437ms/step - accuracy: 0.5261 - loss: 1.1936 - val_accuracy: 0.4003 - val_loss: 1.2939
Epoch 7/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.5564 - loss: 1.1225 - val_accuracy: 0.4739 - val_loss: 1.2111
Epoch 8/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 439ms/step - accuracy: 0.6083 - loss: 0.9944 - val_accuracy: 0.5291 - val_loss: 1.1229
Epoch 9/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.6372 - loss: 0.9307 - val_accuracy: 0.5798 - val_loss: 1.0393
Epoch 10/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 436ms/step - accuracy: 0.6531 - loss: 0.8449 - val_accuracy: 0.6242 - val_loss: 0.9610
Epoch 11/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.6902 - loss: 0.8019 - val_accuracy: 0.6810 - val_loss: 0.8776
Epoch 12/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.7334 - loss: 0.7133 - val_accuracy: 0.7347 - val_loss: 0.7971
Epoch 13/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.7220 - loss: 0.7224 - val_accuracy: 0.7684 - val_loss: 0.7127
Epoch 14/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 450ms/step - accuracy: 0.7425 - loss: 0.6587 - val_accuracy: 0.8083 - val_loss: 0.6327
Epoch 15/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 436ms/step - accuracy: 0.7784 - loss: 0.6221 - val_accuracy: 0.8206 - val_loss: 0.5596
Epoch 16/50
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 437ms/step - accuracy: 0.7907 - loss: 0.5633 - val_accuracy: 0.8466 - val_loss: 0.4942
Epoch 17/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.8052 - loss: 0.5467 - val_accuracy: 0.8758 - val_loss: 0.4399
Epoch 18/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 439ms/step - accuracy: 0.8059 - loss: 0.5117 - val_accuracy: 0.8834 - val_loss: 0.3960
Epoch 19/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 439ms/step - accuracy: 0.8137 - loss: 0.5121 - val_accuracy: 0.8865 - val_loss: 0.3617
Epoch 20/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 440ms/step - accuracy: 0.8320 - loss: 0.4660 - val_accuracy: 0.8942 - val_loss: 0.3337
Epoch 21/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 437ms/step - accuracy: 0.8521 - loss: 0.4218 - val_accuracy: 0.9018 - val_loss: 0.3102
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 437ms/step - accuracy: 0.8414 - loss: 0.4241 - val_accuracy: 0.9080 - val_loss: 0.2902
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.8421 - loss: 0.4045 - val_accuracy: 0.9141 - val_loss: 0.2739
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 437ms/step - accuracy: 0.8511 - loss: 0.3858 - val_accuracy: 0.9141 - val_loss: 0.2601
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.8688 - loss: 0.3553 - val_accuracy: 0.9202 - val_loss: 0.2473
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 450ms/step - accuracy: 0.8821 - loss: 0.3513 - val_accuracy: 0.9264 - val_loss: 0.2362
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.8912 - loss: 0.3200 - val_accuracy: 0.9325 - val_loss: 0.2265
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 438ms/step - accuracy: 0.8909 - loss: 0.3144 - val_accuracy: 0.9340 - val_loss: 0.2175
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 440ms/step - accuracy: 0.8920 - loss: 0.3161 - val_accuracy: 0.9356 - val_loss: 0.2098
```

```
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```




```
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```




```
model.save('efficientnetb0.h5')
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

# Print the results
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - accuracy: 0.9700 - loss: 0.1228
Test Loss: 0.1164
Test Accuracy: 0.9739
```

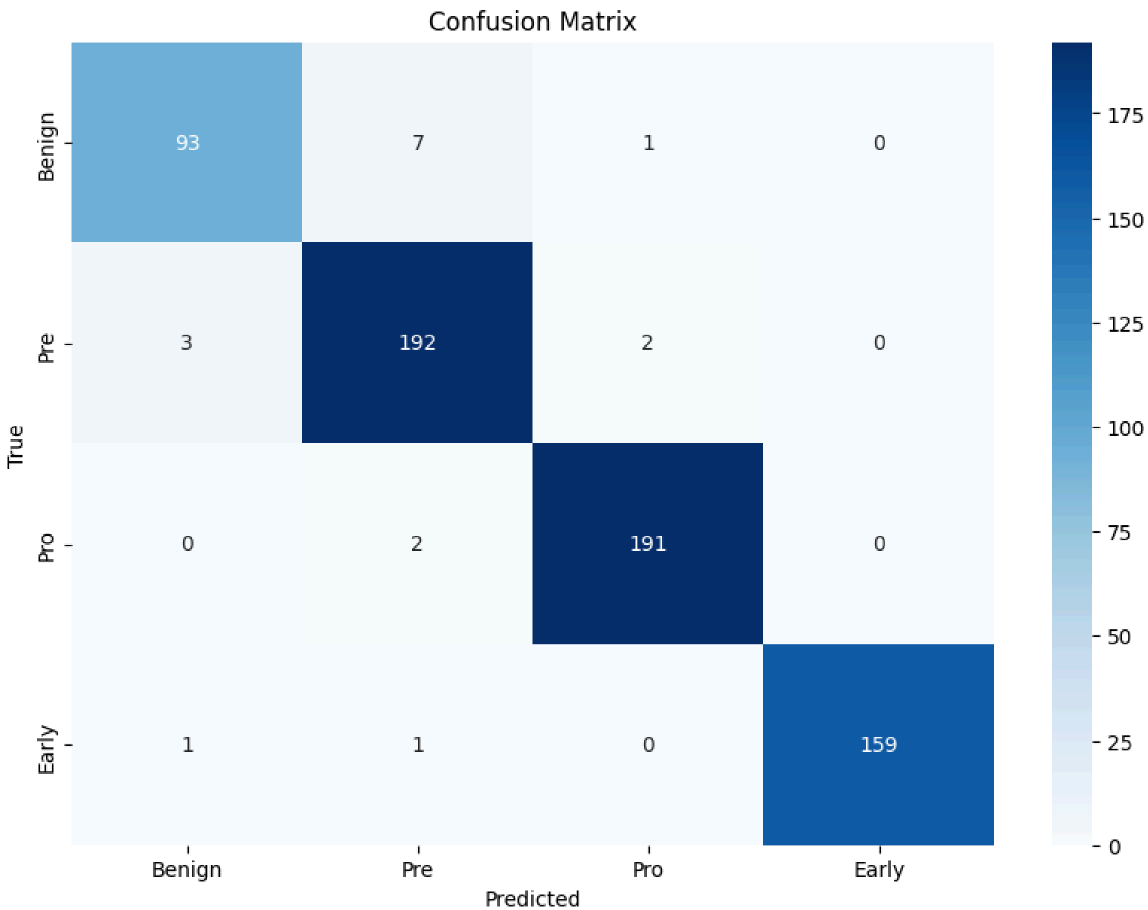
EfficientNetB0 Evaluation Metrics

```
from tensorflow.keras.models import load_model

# Load the model
model = load_model('efficientnetb0.h5')
```

```
from sklearn.metrics import classification_report

# Predict classes for the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Classification report
print(classification_report(y_test, y_pred_classes, target_names=class_names))
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 8s 218ms/step
              precision    recall  f1-score   support

      Benign       0.96      0.92      0.94       101
         Pre       0.95      0.97      0.96       197
         Pro       0.98      0.99      0.99       193
       Early       1.00      0.99      0.99       161

    accuracy                           0.97       652
   macro avg       0.97      0.97      0.97       652
weighted avg       0.97      0.97      0.97       652
```

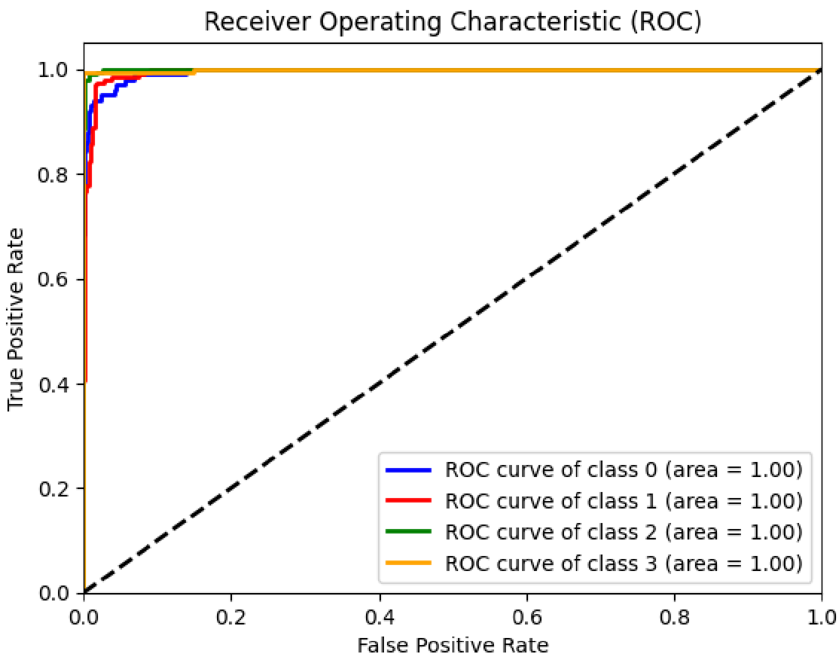
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```




```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Convert y_test to one-hot encoding
y_test_one_hot = label_binarize(y_test, classes=[0, 1, 2, 3])  # Adjust classes as needed

# Predict probabilities
y_prob = model.predict(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_test_one_hot[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Plot ROC curves
plt.figure()
colors = ['blue', 'red', 'green', 'orange']  # Adjust for your classes

for i, color in zip(range(len(class_names)), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step
```



VGG16 MODEL TRAINING

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = Flatten()(x)
```

```
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(class_names), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

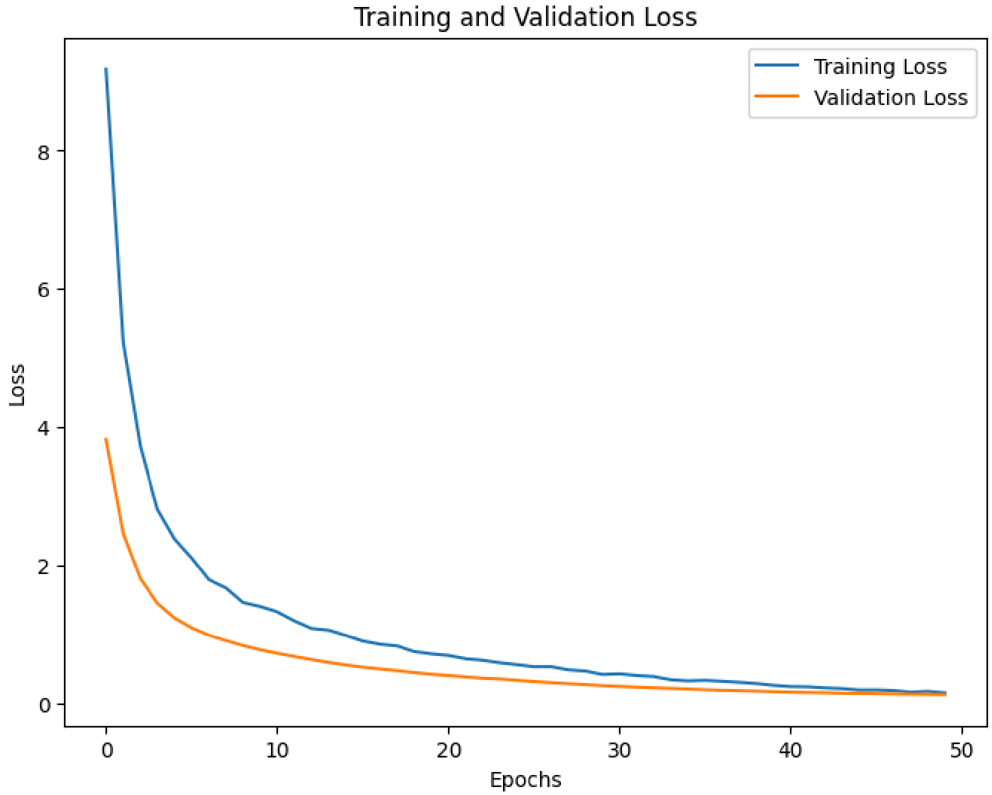
```
model.compile(optimizer=Adam(learning_rate=1e-6),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(train_generator,
                    epochs=50,
                    validation_data=test_generator,
                    callbacks=[early_stopping])
```

```
Epoch 1/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 118s 2s/step - accuracy: 0.2433 - loss: 10.3553 - val_accuracy: 0.2531 - val_loss: 3.8150
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 785ms/step - accuracy: 0.2767 - loss: 5.6396 - val_accuracy: 0.3420 - val_loss: 2.4553
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 784ms/step - accuracy: 0.3113 - loss: 3.9033 - val_accuracy: 0.3865 - val_loss: 1.8018
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 784ms/step - accuracy: 0.3360 - loss: 2.9598 - val_accuracy: 0.4755 - val_loss: 1.4450
Epoch 5/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 786ms/step - accuracy: 0.3861 - loss: 2.4446 - val_accuracy: 0.5276 - val_loss: 1.2320
Epoch 6/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.3988 - loss: 2.1629 - val_accuracy: 0.5752 - val_loss: 1.0888
Epoch 7/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 784ms/step - accuracy: 0.4436 - loss: 1.7856 - val_accuracy: 0.6104 - val_loss: 0.9823
Epoch 8/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.4726 - loss: 1.6692 - val_accuracy: 0.6396 - val_loss: 0.9053
Epoch 9/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.5108 - loss: 1.4713 - val_accuracy: 0.6840 - val_loss: 0.8324
Epoch 10/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 787ms/step - accuracy: 0.5297 - loss: 1.3938 - val_accuracy: 0.7255 - val_loss: 0.7716
Epoch 11/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 785ms/step - accuracy: 0.5348 - loss: 1.3287 - val_accuracy: 0.7500 - val_loss: 0.7213
Epoch 12/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 784ms/step - accuracy: 0.5377 - loss: 1.2622 - val_accuracy: 0.7761 - val_loss: 0.6746
Epoch 13/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.6091 - loss: 1.1170 - val_accuracy: 0.7929 - val_loss: 0.6299
Epoch 14/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 784ms/step - accuracy: 0.6219 - loss: 1.0629 - val_accuracy: 0.8021 - val_loss: 0.5883
Epoch 15/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 785ms/step - accuracy: 0.6232 - loss: 1.0162 - val_accuracy: 0.8206 - val_loss: 0.5531
Epoch 16/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 784ms/step - accuracy: 0.6603 - loss: 0.9106 - val_accuracy: 0.8282 - val_loss: 0.5197
Epoch 17/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 784ms/step - accuracy: 0.6878 - loss: 0.8719 - val_accuracy: 0.8313 - val_loss: 0.4951
```

```
Epoch 18/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.7092 - loss: 0.8145 - val_accuracy: 0.8374 - val_loss: 0.4700
Epoch 19/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.7154 - loss: 0.7653 - val_accuracy: 0.8390 - val_loss: 0.4423
Epoch 20/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 785ms/step - accuracy: 0.7465 - loss: 0.7026 - val_accuracy: 0.8482 - val_loss: 0.4181
Epoch 21/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 784ms/step - accuracy: 0.7449 - loss: 0.7179 - val_accuracy: 0.8589 - val_loss: 0.4004
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.7653 - loss: 0.6559 - val_accuracy: 0.8635 - val_loss: 0.3804
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 785ms/step - accuracy: 0.7823 - loss: 0.6316 - val_accuracy: 0.8666 - val_loss: 0.3608
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 808ms/step - accuracy: 0.7950 - loss: 0.5672 - val_accuracy: 0.8712 - val_loss: 0.3498
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 17s 784ms/step - accuracy: 0.7818 - loss: 0.5750 - val_accuracy: 0.8773 - val_loss: 0.3314
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 786ms/step - accuracy: 0.7996 - loss: 0.5289 - val_accuracy: 0.8850 - val_loss: 0.3135
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.8074 - loss: 0.5243 - val_accuracy: 0.8896 - val_loss: 0.2971
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.8226 - loss: 0.4656 - val_accuracy: 0.8972 - val_loss: 0.2828
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 18s 785ms/step - accuracy: 0.8150 - loss: 0.4681 - val_accuracy: 0.9034 - val_loss: 0.2692
```

```
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```


```
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```




```
model.save('vgg16.h5')
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

# Print the results
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 65ms/step - accuracy: 0.9520 - loss: 0.1488
Test Loss: 0.1262
Test Accuracy: 0.9632
```

VGG16 Evaluation

```
from tensorflow.keras.models import load_model

# Load the model
model = load_model('vgg16.h5')
```

```
from sklearn.metrics import classification_report

# Predict classes for the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Classification report
print(classification_report(y_test, y_pred_classes, target_names=class_names))
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step
              precision    recall  f1-score   support

      Benign       0.90      0.91      0.91       101
         Pre       0.94      0.94      0.94       197
         Pro       0.98      0.98      0.98       193
       Early       1.00      0.99      1.00       161

    accuracy                           0.96       652
   macro avg       0.96      0.96      0.96       652
weighted avg       0.96      0.96      0.96       652
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
```